Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.М04-мм

Обзор алгоритма для поиска зависимостей включения подобия Sawfish

Чижов Антон Игоревич

Отчёт по производственной практике
в форме «Производственное задание»

Научный руководитель:
ассистент кафедры информационно-аналитических систем Г. А. Чернышев

Санкт-Петербург
2025

Оглавление

Введение

Профилирование данных [1] является комплексным аналитическим процессом, цель которого заключается в выделении ключевых метаданных. Наиболее интересным для изучения является наукоёмкое профилирование, связанное с поиском зависимостей в данных. Наиболее известный пример — функциональные зависимости, которые являются ограничениями целостности в базах данных [3]. Данные зависимости, например, могут использоваться для проверки того, находится ли отношение в третьей нормальной форме [3].

Хотя ограничения целостности, такие как функциональные зависимости, успешно описывают семантику данных, некоторые данные в базе данных могут не соответствовать этим ограничениям. Одной из причин является тот факт, что данные могут поступать из различных независимых источников. В отличие от традиционных функциональных зависимостей, зависимости включения [7,8] (Inclusion Dependency, IND) могут использовать разные отношения для проверки корректности данных.

Неформально, между двумя наборами колонок из двух (необязательно разных) отношений существует зависимость включения, если для каждого кортежа из первого набора атрибутов в первом отношении существует такой же кортеж из второго набора атрибутов во втором отношении.

Аналогично функциональным зависимостям, для которых определено множество зависимостей, ключевой идеей которых является задание метрики для подсчёта ошибки, для зависимостей включения определяются приближённые зависимости включения [5,10] и зависимости включения подобия (Similarity Inclusion Dependency, SIND) [9]. Данные зависимости можно применять для большего набора задач, например для обнаружения отсутствующих ограничений и проблем с целостностью данных.

Desbordante [6] представляет собой научно-ориентированный профилировщик данных с открытым исходным кодом[1]. Создание Desbordante

[1] https://github.com/Desbordante/desbordante-core

было мотивировано недостатками существующей платформы Metanome [4]. В Desbordante в качестве основного языка используется `C++`, а в Metanome — `Java`, из-за чего Metanome имеет не самую оптимальную производительность. Desbordante включает в себя алгоритмы для поиска различных примитивов и также предоставляет соответствующие интерфейсы для пользователей.

В данный момент Desbordante активно развивается, имеет поддержку AIND и требует расширения набора поддерживаемых примитивов, в том числе поддержку SIND. Данная работа посвящена исследованию зависимостей включения подобия, в частности анализу алгоритма `Sawfish` [9] для их поиска.

Постановка задачи

Целью данной работы является изучение алгоритма для поиска зависимостей включения подобия для расширения возможностей анализа данных. Для достижения этой цели были поставлены следующие задачи.

- Выполнить обзор предметной области и сравнение расширений зависимостей включения.
- Выполнить обзор алгоритма для поиска зависимостей включения подобия Sawfish.

1 Обзор предметной области

В данной секции вводятся базовые определения в предметной области зависимостей включения и их расширений.

Определение 1. ***Зависимость включения (Inclusion Dependency, IND)*** *определяется следующим образом:*

$$IND : R_1[X] \subseteq R_2[Y],$$

где:

1. *X и Y это наборы атрибутов из отношений R_1 и R_2 соответственно (X — зависимые, а Y — ссылочные атрибуты);*

2. $\forall t_1 \in \pi_X(R_1)\ \exists t_2 \in \pi_Y(R_2) : t_1 = t_2$.

   Другими словами, для каждого зависимого значения существует такое же ссылочное значение.

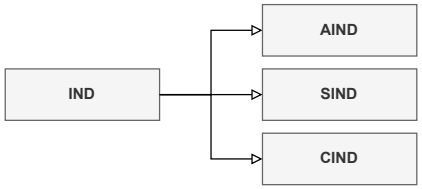

Рис. 1: Расширения зависимостей включения.

Точные зависимости включения являются хорошо изученным типом зависимостей. Однако такие зависимости имеют достаточно узкий круг применения, в то время как на практике удобнее вводить некоторые их расширения. Известные на данный момент возможные расширения представлены на Рис. 1.

Во-первых, реальные данные зачастую содержат ошибки, из-за чего нельзя обнаружить точные зависимости. Для работы с такими данными изначально были введены приближённые зависимости включения (AIND), которые могут выполняться с некоторой ошибкой.

Определение 2. ***Приближённая зависимость включения [10] (Approximate Inclusion Dependency, AIND)*** *определяется следующим образом:*

$$AIND : R_i(X) \ \subseteq_\epsilon R_j(Y),$$

где пороговое значение ϵ задаёт максимальное значение ошибки для функции g'_3, т.е. $g'_3(R_i(X) \subseteq R_j(Y)) \leq \epsilon$.

Пусть d это база данных над схемой базы данных R и r_i, $r_j \in d$ для соответствующих отношений R_i, $R_j \in \boldsymbol{R}$. Тогда функция g'_3 вычисляет соотношение минимального количества кортежей, которые необходимо удалить из r, чтобы $R_i(X) \subseteq R_j(Y)$ выполнялась над r.

$$g'_3(R_i(X) \subseteq R_j(Y)) = \\ = 1 - \frac{max\{|\pi_X(r')| : r' \subseteq r_i \ u \ (d - \{r_i\}) \cup \{r'\} \models R_i(X) \subseteq R_j(Y)\}}{|\pi_X(r_i)|},$$

где r' это подмножество кортежей в r_i, которые не нарушают зависимость.

Приближённые зависимости являются наиболее известным среди расширений зависимостей включения. Они выполняются на всей выборке данных и показывают, что зависимость выполняется с определённой вероятностью. При этом приближённые зависимости не дают представления о самих ошибках в данных.

Условные зависимости включения определяются подобно условным функциональным зависимостям, т.е. зависимость должна выполняться на части отношения вместо всего отношения.

Определение 3. ***Условная зависимость включения [2] (Conditional Inclusion Dependency, CIND)*** *определяется следующим образом:*

$$CIND : R_1[X; X_p] \subseteq R_2[Y; Y_p], t_p,$$

где:

1. *X, X_p и Y, Y_p являются наборами атрибутов отношений R_1 и R_2, соответственно, так что X и X_p не пересекаются (nil —*

пустое множество).

2. $R_1(X) \subseteq R_2(Y)$ *— стандартная зависимость включения, называемая вложенной в условную зависимость включения.*

3. t_p *— это шаблонный кортеж с атрибутами* X, X_p *и* Y, Y_p, *где для каждого* $A \in X \cup X_p \cup Y \cup Y_p$, $t_p[A]$ *либо является конкретной константой* $a \in dom(A)$, *либо является анонимной переменной* _ , *принимающей значения из* $dom(A)$.

В случае условных зависимостей включения используется отличный подход от приближённых зависимостей включения, поскольку анализируется только часть отношения, в то время как зачастую для изучения более интересны зависимости, применимые ко всему отношению. При этом условные зависимости также не объясняют, какие ошибки приводят к нарушению стандартных зависимостей включения.

Типичным решением данной проблемы является использование меры подобия для того, чтобы подсчитывать небольшие ошибки, такие как опечатки и отличия в форматировании. В статье [9] вводится определение для зависимостей включения подобия, при котором точная зависимость включения "ослабляется", требуя существования только достаточно похожих значений.

Определение 4. ***Зависимость включения подобия [9] (Similarity Inclusion Dependency , SIND)*** *определяется следующим образом:*

$$SIND : R_1[X] \subsetneq_\sigma R_2[Y],$$

где:

1. X *и* Y *являются наборами атрибутов отношений* R_1 *и* R_2, *соответственно.*

2. $\forall t_1 \in \pi_X(R_1)\ \exists t_2 \in \pi_Y(R_2) : t_1 \approx_\sigma t_2$

   Другими словами, для каждого зависимого значения существует подобное ссылочное значение с учётом меры и порога ошибки.

Для зависимостей включения подобия могут использоваться разные меры подобия. Далее будут рассматриваться меры, которые определяется с помощью расстояния Левенштейна (режим редактирования, ED mode) и с помощью коэффициентов Жаккара (режим токенов, JAC mode).

Сравнение IND, AIND и SIND. Для сравнения различных зависимостей включений рассмотрим следующие таблицы:

name	**instructor**
Machine Learning	Dr. Smith
Data Structures	Dr. Johnson
Algorithms	Dr. Brown
Databases	Dr. Lee

Таблица 1: course

student_id	**course_name**
1024	Machine Learning
1031	Machine Learning
1045	Data Structures
1077	Algorithm_s
1102	Databsses
1140	Data Structures

Таблица 2: enrollment

В таблица 1 определено соответствие между курсами и преподавателями, которые ведут эти курсы. А в таблице 2 содержится информация о курсах, которые выбрали студенты. При этом часть студентов допустила опечатки в названиях курсов (ошибки подчёркнуты в таблице).

Предположим, что мы хотим исследовать следующую зависимость:

$$enrollment[course\_name] \subseteq course[name]$$

IND. Среди зависимых значений встречаются опечатки, из-за чего зависимость не выполняется. В таких случаях использование точных зависимостей включения не помогает анализировать данные.

AIND. В случае приближённых зависимостей необходимо подсчитать ошибку, с которой зависимость удерживается. Множество уникальных зависимых значений равняется шести, а количество неправильных уникальных значения равняется трём. Соответственно ошибка равняется $\frac{3}{6} = 0.5$. Соответственно для того, чтобы обнаружить данную зависимость необходимо использовать низкий порог ошибки. Однако использование настолько низкого порога ошибки на практике ведёт к большому количеству лишних зависимостей, которые не дают никакой

дополнительной информации о данных, а скорее являются случайностью. Проблема заключается в том, что приближённые зависимости включения не учитывают особенности ошибок и не дают никакой дополнительной информации о них.

SIND. Для зависимости включения подобия будем использовать расстояние Левенштейна. Можно заметить, что в зависимой части в каждом ошибочном значении содержится максимум одна опечатка. Соответственно данная зависимость будет найдена с минимальным порогом ошибки, т.е. при $\sigma = 1$. Для работы с такими данными удобно использовать зависимости включения подобия с различными мерами схожести в зависимости от природы данных.

2 Алгоритм Sawfish

Алгоритм `Sawfish` (**S**imilarity **AW**are **F**inder of **I**nclusion dependencies via a **S**egmented **H**ashindex) [9] является единственным известным на данный момент алгоритмом для поиска унарных зависимостей включения подобия.

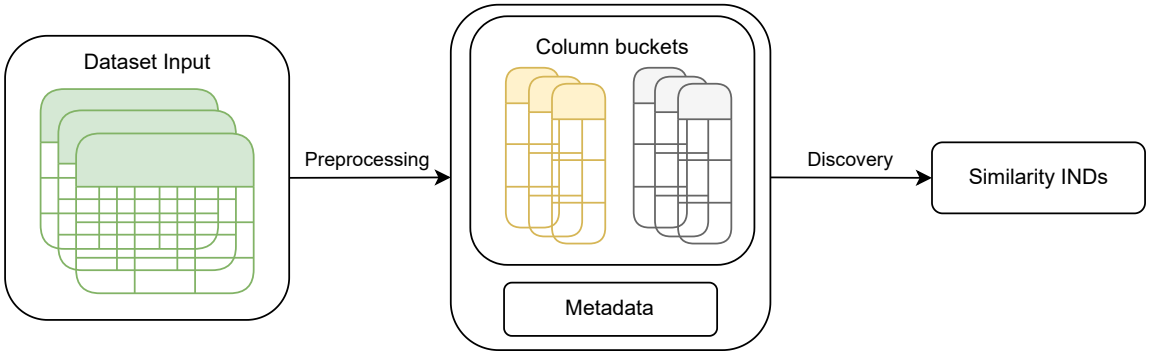

Рис. 2: Алгоритм Sawfish

На рисунке 2 показаны стадии работы алгоритма. `Sawfish` предварительно обрабатывает входные данные, генерирует метаданные, затем отмечает кандидатов зависимостей включения подобия, требующие последующей проверки. Затем он выполняет проверку кандидатов с использованием инвертированного индекса, который используется для выявления подобных строк.

Данный алгоритм может использовать любые меры подобия, которые удовлетворяют следующим свойствам:

1. иметь возможность отсекать пары значений на основе упорядоченной числовой характеристики;

2. иметь возможность отсекать пары значений на основе неравенства их подстрок.

Второе свойство необходимо для создания инвертированного индекса, где подстроки указывают на свои родительские строки. А если две строки не имеют общих подстрок, то они считаются несхожими.

Среди других мер схожести можно выделить расстояние Хэмминга, косинусное сходство (Cosine similarity) и коэффициент Сёренсена (Dice

similarity).

Далее в данной секции рассматривается этапы работы алгоритма.

2.1 Обработка данных

На первом этапе входные данные преобразуются в специальный формат, а также извлекаются дополнительные метаданные.

Для каждого столбца все уникальные значения группируются по их длине. В случае расстояния Левенштейна длина измеряется в количестве символов в строке, а в случае коэффициента Жаккара — в количестве токенов в строке. При этом отбрасываются колонки, которые содержат значения, длина которых более 50 символов или более 10 токенов, в случае работы с токенами. Также, для каждого столбца собирается информация о максимальной и минимальной длине его значений.

Алгоритмы для работы с зависимостями включения могут использоваться для работы с большими данными, из-за чего могут возникать проблемы с недостатком оперативной памяти. Для таких случаев используется контроль памяти и выгрузка колонок значений на диск. Если на этапе обработки входных данных алгоритм достигает ограничения по памяти, то алгоритм выгружает самый большой столбец на диск.

Данные на диске хранятся в отдельных файлах (файлы группируются по индексу колонки и длине значений) для последующего чтения без необходимости повторного чтения всей таблицы. После полной обработки входных данных проводится удаление дупликатов выгруженных на диск данных, так как в них могут присутствовать дублирующиеся данные из-за раннего сброса буферов.

Также, на этапе обработки входных данных происходит отсеивание неподходящих кандидатов. Например, отсеиваются кандидаты с пустыми столбцами, рефлексивные зависимости. Также, в случае расстояния Левенштейна исключаются кандидаты, в которых все значения короче порогового значения. Отбрасываться также могут кандидаты, которые не могут быть истинными из-за разницы в длинах значений.

После отсеивания оставшиеся кандидаты упорядочиваются, назначая каждой ссылочной колонке все потенциально зависимые от неё колонки.

2.2 Поиск зависимостей

Основной принцип `Sawfish` заключается в том, что для отсечения кандидата `SIND` достаточно найти всего один контрпример. В отличие от зависимостей включения, в случае зависимостей включения подобия для каждого зависимого значения должно существовать хотя бы одно похожее значения в ссылочном столбце.

Для уменьшения числа проверок, `Sawfish` использует инвертированный индекс. Для ссылочного столбца строится инвертированный индекс, после чего для каждого значения из зависимого столбца выполняется поиск подобного значения по данному индексу.

2.2.1 Инвертированный индекс

Инвертированный индекс хранит отображение уникальных подстрок на их исходные строки. В зависимости от режима работы данные хранятся в разном виде.

- В режиме работы с токенами индекс сопоставляет токены и их исходные строки.
- В режиме работы с расстоянием Левенштейна каждая строка разбивается на сегменты, а индекс сопоставляет сегменты и их исходные строки.

В случае работы с токенами алгоритм построения индекса очевиден, поэтому далее более подробно рассмотрим построение индекса в случае работы с расстоянием Левенштейна.

Сначала генерируются позиции сегментов, при этом позиции достаточно вычислить один раз для конкретного значения длины. Длина

сегментов делается практически одинаковой. Если есть остаток при делении длины строки на количество сегментов, то более короткие сегменты размещаются в начале.

Подстроки извлекаются на основе сгенерированных позиций, при этом каждая подстрока сопоставляется со своей исходной строкой. Построение индекса должно происходить последовательно для возможности использования скользящего окна по длинам. Поэтому наборы сегментов хранятся отдельно и независимо друг от друга.

Важной особенностью инвертированного индекса является возможность использования скользящего окна по длинам: достаточно сравнивать только зависимые значения со значениями внутри индекса в пределах допустимого интервала. Итерация по длинам значений начинается с самых длинных к самым коротким. При переходе от одной длины к другой происходит удаление ненужного индекса и происходит загрузка нового.

Благодаря скользящему окну нет необходимости строить весь индекс (если столбец перестаёт быть ссылочным, то индексация прекращается досрочно), а индексы занимают меньше памяти.

2.2.2 Проверка зависимых значений

Основное различие между разными режимами работы алгоритма осуществляется в том, как выполняется проверка зависимых значений из зависимых столбцов.

Режим работы с расстоянием Левенштейна (ED mode). Для проверки кандидатов создаются подстроки, которые могут соответствовать сегментам из инвертированного индекса. Для того, чтобы две строчки x и y были подобны, как минимум одна подстрока y должна совпасть с сегментом x. Поскольку инвертированный индекс основан на сегментах, необходимо сгенерировать все подстроки, которые могут соответствовать сегменту. При этом используются техники для уменьшения количества сравнений, избегая необязательные подстроки.

При обращении к индексу происходит проверка, содержится ли в нём хотя бы одно значение, которое является похожим на проверяемое

значение из зависимого столбца. Поскольку строки могут иметь разную длину, то перебираются все возможные различия в длине (ld) между зависимым значением и индексируемыми значениями (с учётом допустимого порога расстояния между значениями).

Для каждой возможной длины, соответствующей ld, сначала генерируется множество подстрок, которые могут совпадать с сегментами из индекса. Для того, чтобы избежать проверки всех возможных подстрок строки из зависимого столбца, применяется оптимизированная стратегия.

Поскольку инвертированный индекс разделён по длинам значений, мы выбираем соответствующий индекс в зависимости от текущей длины проверяемого значения. После выбора нужного индекса происходит сравнение подстрок с соответствующими сегментами из индекса.

После получения потенциально похожих ар строк из инвертированного индекса мы проверяем их схожесть, вычисляя точное расстояние редактирования. Вместо полного динамического программирования используется оптимизация, позволяющая избежать вычисления всех элементов матрицы. Поскольку нам необходимо выяснить только факт схожести, вычисление прерывается, если дистанция превысила максимальный порог схожести. Во-вторых, учитывая найденные совпадающие сегменты, вычисление разбивается на две части — левую и правую от совпадения, что позволяет задать более точное пороговое значение.

Режим работы с коэффициентом Жаккара (JAC mode). В случае работы с токенами метод проверки схожести значительно проще, чем для расстояния Левенштейна. Зная количество токенов в инвертированном индексе и в зависимом значении, необходимо вычислить минимальное число совпадающих токенов (T), необходимое для признания схожести. Для идентификации похожих строк используется метод сканирования. После определения порога T необходимо подсчитать количество точных совпадений между токенами зависимого значения и у каждого элемента индекса. Если для какого-либо элемента найдено не менее T совпадений, то это зависимое значение считается достаточно похожим.

2.2.3 Поиск SIND

В данной части используются описанные ранее этапы, объединяя их в единый процесс. Для каждой ссылочной колонки создаётся инвертированный индекс, который затем используется для проверки всех возможных зависимых колонок.

Процесс поиска зависимостей включения подобия заключается в запрашивании схожих значений у индекса для каждого зависимого значения. Если мы не нашли совпадающих подстрок, то кандидат сразу отбрасывается. В противном случае найденные совпадения дополнительно проверяются с помощью меры схожести. Если хотя бы одно из них соответствует пороговому значению, переходят к следующему значению. После обработки всех зависимых значений колонка остаётся в списке кандидатов только в случае, если зависимость выполняется.

Для повышения эффективности вместо поочерёдной обработки колонок, что не всегда оптимально, в память может загружаться сразу несколько индексов, если позволяет объём оперативной памяти. Это позволяет избегать повторной загрузки данных.

Заключение

По итогам работы получены следующие результаты.

- Выполнен обзор предметной области зависимостей включения и выполнено сравнение расширений зависимостей включения.
- Выполнен обзор алгоритма для поиска зависимостей включения подобия Sawfish.

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, Naumann Felix. Profiling relational data: a survey // The VLDB Journal. — 2015. — aug. — Vol. 24, no. 4. — P. 557–581. — URL: http://dx.doi.org/10.1007/s00778-015-0389-y.

[2] Bravo Loreto, Fan Wenfei, Ma Shuai. Extending Dependencies with Conditions. — 2007. — 01. — P. 243–254.

[3] Codd E. F. Further Normalization of the Data Base Relational Model // Research Report / RJ / IBM / San Jose, California. — 1971. — Vol. RJ909. — URL: https://api.semanticscholar.org/CorpusID:45071523.

[4] Data profiling with metanome / Thorsten Papenbrock, Tanja Bergmann, Moritz Finke et al. // Proceedings of the VLDB Endowment. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — URL: http://dx.doi.org/10.14778/2824032.2824086.

[5] De Marchi Fabien, Petit Jean-Marc. Approximating a Set of Approximate Inclusion Dependencies. — 2005. — 01. — P. 633–640.

[6] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms / Maxim Strutovskiy, Nikita Bobrov, Kirill Smirnov, George Chernishev // 2021 29th Conference of Open Innovations Association (FRUCT). — IEEE, 2021. — may. — URL: http://dx.doi.org/10.23919/FRUCT52173.2021.9435469.

[7] Fagin Ronald. Horn clauses and database dependencies (Extended Abstract) // Symposium on the Theory of Computing. — 1980. — URL: https://api.semanticscholar.org/CorpusID:6285434.

[8] Fagin Ronald. A Normal Form for Relational Databases That is Based on Domains and Keys // ACM Trans. Database Syst. — 1981. — sep. — Vol. 6, no. 3. — P. 387–415. — URL: https://doi.org/10.1145/319587.319592.

[9] Kaminsky Youri, Pena Eduardo, Naumann Felix. Discovering Similarity Inclusion Dependencies // Proceedings of the ACM on Management of Data. — 2023. — 05. — Vol. 1. — P. 1–24.

[10] Lopes Stéphane, Petit Jean-Marc, Toumani Farouk. Discovering Interesting Inclusion Dependencies: Application to Logical Database Tuning // Inf. Syst. — 2002. — mar. — Vol. 27, no. 1. — P. 1–19. — URL: https://doi.org/10.1016/S0306-4379(01)00027-8.